REMARKS

Claims 1-40 are pending, with claims 1, 16, 24, and 32 being independent. Reconsideration and allowance of the above-referenced application are respectfully requested in view of the following remarks.

Claims 1-40 stand rejected under 35 U.S.C. 103(a) as allegedly being unpatentable over D'Arlach et al. (US 6,026,433) in view of Szabo (US 6,868,525). Claims 1-40 also stand rejected under 35 U.S.C. 103(a) as allegedly being unpatentable over D'Arlach et al. in view of Dilworth et al. (WO 00/51018). These rejections are respectfully traversed, and reconsideration and allowance of the present application are respectfully requested in view of the remarks herein.

The presently claimed subject matter is generally directed to client-side modification of electronic documents by server-generated instructions in a client-server environment. As exemplified in claim 32, the claimed operations include: "receiving a request from a client; producing data corresponding to the client request; generating instructions for an electronic document having a predetermined format that defines an appearance of the electronic document independent of a device used to present the electronic document, the generated instructions specifying one or more operations to modify the predetermined format at the client to accommodate the produced data, the generated instructions to be performed at the client to effect the one or more operations; and transmitting the produced data and the generated instructions to the client." (Emphasis added.)

D'Arlach describes a "method for creating and editing a Web site in a client-server computer network using customizable templates." (*See* D'Arlach at Abstract.) The Official Action acknowledges that D'Arlach does not describe an electronic document having a predetermined format that defines an appearance of the electronic document independent of a device used to present the electronic document, and relies on Szabo and Dilworth for this feature.

Szabo describes a "human user computer interface system, providing a graphic representation of a hierarchy populated with naturally classified objects, having included therein at least one associated object having a distinct classification." (*See* Szabo at Abstract.) The Official Action contends that Szabo teaches the use of XML (Extensible Markup Language) as

an electronic document having a predetermined format that defines an appearance of the electronic document independent of a device used to present the electronic document. Szabo does not in any way teach or suggest this. In particular, the cited portion of Szabo (col. 2, lines 15-30) cannot support the Office Action's assertion that, "an XML document will appear the same no matter what platform the OS is", for this portion of Szabo says nothing about the appearance of an XML document on different platforms, and in fact, an XML document will appear very differently on different platforms depending on the software used to open and display the XML document.

Moreover, the statement that, "an XML document will appear the same no matter what platform the OS is" does not actually address the claimed feature. The claim recites an electronic document having a predetermined format that defines an appearance of the electronic document independent of a device used to present the electronic document. The format of XML does not define document appearance at all. XML is by definition a general-purpose markup language that can be rendered differently in different Web browsers. (*See e.g.*, http://en.wikipedia.org/wiki/XML#Displaying_XML_on_the_web; a copy of which is included with this response as a ten page attachment.) Thus, the claimed feature has not actually been addressed by the current Office Action.

As described in paragraphs 9 and 10 of the present application, the claimed electronic document is generally a type of document that allows a publisher to control the look and feel of the document as seen by an end user, including the specific physical page or pages on which information appears when printed or displayed. Thus, the format should generally support and preserve all visual formatting features (e.g., fonts, graphics, color, etc.) of any source document, regardless of the source computer platform and/or software application used to create the source document. The ability to control final appearance, or look-and-feel, of an electronic document as viewed by a reader can be a critical branding issue for businesses and other publishing organizations, and is particularly useful when available across various computer platforms. An example of this type of document format is the PORTABLE DOCUMENT FORMAT (PDF)

developed by Adobe Systems, Inc. of San Jose, California. XML is in no way this type of document, and Szabo does not in any way suggest such.

Dilworth describes an on-line editor that "operates as though embedded in a Java-enabled Web browser which enables a user to edit information in a word processor environment. The editor supports most of the HTML [Hypertext Markup Language] formats, including support for graphics. Because the editor in the preferred embodiment is written in the Java programming language, it is cross-platform compatible with most Web browsers. A user is thus required to have a Java-enabled Web browser to access and interactively edit content while on-line to the site. The information generated by the user with the Java editor is stored automatically on the site server when the user clicks on the save button. Thus, there is no need for a site administrator or for any extensive HTML language knowledge, knowledge of Internet protocols, or extensive computer knowledge." (*See* Dilworth at Abstract.)

The Official Action cites this Abstract of Dilworth as teaching an electronic document, having a predetermined format that defines an appearance of the electronic document independent of a device used to present the electronic document. This cannot be supported for two reasons. First, the device-independent aspect of Dilworth is the editor software itself (due to its being written in the Java programming language) not a predetermined format that defines an appearance of an electronic document as claimed. Second, Dilworth describes a Java editor operating as though embedded in a Web browser to provide a word processor environment to a user, and the editor supports "most of the HTML formats". (*See* Dilworth at page 1, lines 14-17, and page 3, lines 20-25.) Aside from this reference to HTML formats, Dilworth makes no further mention of the format of electronic documents in the system.

An HTML document generally has no ability to specify which portions of the document will appear on which physical pages when printed. In fact, a publisher of an HTML document has no final control over how the document will appear to an end user, because presentation of an HTML document, either by display on a monitor or by printing, is determined by the Web browser, which interprets the HTML tags. Thus, the reference to HTML formats by Dilworth does not describe the claimed electronic document having a predetermined format that defines an

appearance of the electronic document independent of a device used to present the electronic document.

For all of the above reasons, and due to insufficient motivation to combine the references as suggested, a prima facie case of obviousness has not been established for any of claims 1-40.

In addition, D'Arlach does not describe, as is suggested in the Official Action, generating instructions to modify an electronic document, the generated instructions specifying one or more operations to modify the electronic document at the client to accommodate the produced data, and the generated instructions to be performed at the client to effect the one or more operations. The support given by the Official Action for these claimed features is D'Arlach's Abstract and FIGs. 4 and 8-14. However, nowhere does D'Arlach's Abstract or the referenced figures describe or show generation of instructions to be performed at the client to effect the one or more operations. D'Arlach clearly describes a system in which a client sends requests to a server, which generates documents to be sent to the client in the course of building a Web site using customizable templates. Copies of the template documents are changed in response to user requests, but D'Arlach makes very clear that all such changes occur on the server computer, not at the client. (*See* D'Arlach at col. 5, lines 26-32.) Thus, D'Arlach actually teaches away from generating instructions to send to a client to modify an electronic document at the client.

As briefly summarized in paragraph 12 of the present application, "Client-side modification of electronic documents by server-generated instructions in a client-server environment enables dynamic modification of formatting information in an electronic document to accommodate new data received from the server. A server generates machine instructions to send to a client along with new data for an electronic document. The generated instructions modify the electronic document at the client to accommodate the new data."

In view of the above, it should now be clear that the art of record fails to teach or suggest the claimed subject matter. For all of the above reasons, a prima facie case of obviousness has not been established for any of claims 1-40, and all of the rejections of claims 1-40 should be withdrawn.

In addition, independent claim 24 and the various dependent claims include additional recitations not taught by the art of record and which provide independent grounds for the patentability of these claims. For example, claims 6, 21, 24, and 37 recite the feature, "the generated instructions further comprise at least one tag indicating an order in which the produced data is to be imported into the electronic document and the instructions are to be performed." The Official Action cites D'Arlach's Abstract and FIGs. 4, 6, and 7-14 for this feature without any explanation. D'Arlach has been thoroughly reviewed and there is no description or figure in D'Arlach that mentions or shows a tag relating to an order in which to import produced data into an electronic document and to perform instructions that modify the document to accommodate the produced data.

Claims 7, 22, 25, 26, 27, and 38 recite specific types of tags that (1) indicate the produced data is to be imported into the electronic document before the instructions are performed, (2) indicate the produced data is to be imported into the electronic document after the instructions are performed, or (3) indicate that at least a portion of the generated instructions are to be inserted into the electronic document. The cited portions of D'Arlach (FIGs. 4-13) are unrelated to this claimed subject matter. FIG. 4 illustrates a simplified database structure; FIG. 5 illustrates a flow diagram of major steps taken in creating a Web site or editing a site based on style templates; FIG. 6 illustrates a flow diagram of basic steps involved in editing an element and its attributes in a page of a site; and FIGs. 7-13 illustrate various user's perspectives of sample display screens. (*See* D'Arlach at col. 2, line 65 to col. 3, line 25.) Thus, the rejection of these claims cannot be supported by D'Arlach.

With respect to claims 4, 19, 29, and 35, the cited portion of D'Arlach (col. 5, lines 46-55) describes a server that edits a working database copy of a Web site being built, in response to user interaction with HTML forms in a Web browser, where the original databases remain in tact to be used again in creating new sites. In other words, the server creates a copy of a Web site template and then modifies that copy in response to user instructions. This does not describe, "adding information to the electronic document without changing pre-existing information in the electronic document", as claimed. To the contrary, a copy of the original

template is used in D'Arlach precisely because changes are going to be made that overwrite pre-existing information; thus the pre-existing information is retained in the original document, which is not modified, for later use.

With respect to claims 5, 20, 30, and 36, the cited portion of D'Arlach (FIG. 4) illustrates a database structure, not a script. (See D'Arlach at col. 2, lines 65-67.) With respect to claims 9 and 40, the cited portions of D'Arlach (FIGs. 4 and 7-13) illustrate a database structure and various user's perspectives of sample display screens. Overlay of visual objects in an electronic document is neither described nor shown in D'Arlach.

In general, the Official Action's extensive use of bare citations to the figures of D'Arlach without corresponding references to the detailed description is objected to as improper, since the description corresponding to the referenced figures makes clear that D'Arlach neither expressly nor implicitly teaches the presently claimed subject matter. As described in connection with FIG. 4 of D'Arlach:

> According to the present embodiment, style templates are stored in the server computer as database files and consist of a set of objects or elements stored in the database. To allow creating a new Web site, a CGI [Common Gateway Interface] program first makes a copy of an existing template in the server computer. The user then customizes or edits the working copy of the template, which is the user's site, through a series of forms displayed by a browser in the client computer. After making desired changes to the site, the user may publish it as a new Web site in case of creating or as a new version of a Web site in case of editing a site. The original templates and their databases are preserved at all times.
>
> A user selects a style template upon which his or her Web site will be based on. In the present embodiment, when a user selects a template to create a site, the template is duplicated in the server computer. All changes to the copy of the template occur on the server computer. When the template is published as a Web site, the database is used to generate a set of Web pages that make up the new site.

(*See* D'Arlach at col. 5, lines 14-32.) The CGI program is part of the server, and the server interacts with the client in a traditional manner (the server receives requests from the client, and in response, generates a document and delivers such to the client). (*See* D'Arlach at col. 4, lines 8-51.) The description of FIGs. 5-14 makes clear that a traditional client-server page delivery model is being used (as described in connection with FIGs. 2-3) to enable a user to build a customized Web site on a server. (*See* D'Arlach at col. 5, line 33 to col. 10, line 3.)

Therefore, D'Arlach clearly describes a system in which a client sends requests to a server, which generates documents to be sent to the client in the course of building a Web site using customizable templates. D'Arlach does not teach or suggest client-side modification of electronic documents as recited in the independent claims, nor does D'Arlach teach or suggest the details of such modification of an electronic document as recited in the dependent claims. Thus, independent claims 1, 16, 24, and 32 are in condition for allowance. Moreover, dependent claims are 2-15, 17-23, 25-31, and 33-40 are allowable for at least the same reasons as their corresponding independent claims, and based on additional recitations they contain.

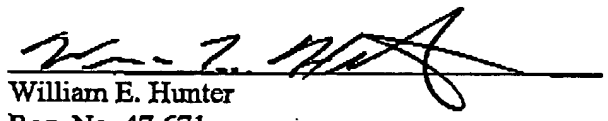
The foregoing comments made with respect to the positions taken by the Examiner are not to be construed as acquiescence with other positions of the Examiner that have not been explicitly contested. Accordingly, the above arguments for patentability of a claim should not be construed as implying that there are not other valid reasons for patentability of that claim or other claims.

A formal notice of allowance is respectfully requested. In the absence of such, a telephone interview with the Examiner and the Examiner's supervisor is respectfully requested to discuss the independent claims 1, 16, 24, and 32, and the dependent claims 4, 6, 7, 9, 15, 19, 21, 22, 25-29, 35, 37, 38, and 40, in view of D'Arlach.

No fees are due at this time. Please apply any necessary charges or credits to deposit account 06-1050.

Respectfully submitted,

Date: September 19, 2005



William E. Hunter
Reg. No. 47,671

Fish & Richardson P.C.
Customer Number: 021876
12390 El Camino Real
San Diego, California 92130
Telephone: (858) 678-5070
Facsimile: (858) 678-5099

10538598.doc

XML

From Wikipedia, the free encyclopedia.

The **Extensible Markup Language (XML)** is a W3C-recommended general-purpose markup language for creating special-purpose markup languages. It is a simplified subset of SGML, capable of describing many different kinds of data. Its primary purpose is to facilitate the sharing of data across different systems, particularly systems connected via the Internet. Languages based on XML (for example, RDF, RSS, MathML, XHTML, SVG, and cXML) are defined in a formal way, allowing programs to modify and validate documents in these languages without prior knowledge of their form.

Contents

History

In the mid-1990s, some practitioners of SGML had gained experience with the then-new World Wide Web, and believed that SGML offered solutions to some of the problems the Web was likely to face as it grew. Jon Bosak argued that the W3C should sponsor an "SGML on the Web" activity. After some resistance, he was authorized to launch that activity in mid-1996, albeit with little involvement by or support from the W3C leadership. Bosak was well-connected in the small community of people who had experience both in SGML and the Web. He received active support in his efforts from Microsoft.

XML was designed by an eleven-member Working Group supported by an (approximately) 150-member Interest Group. Technical debate took place on the Interest Group mailing list, and issues were resolved by consensus, or when that failed, majority vote, of the Working Group. James Clark served as Technical Lead of the Working Group, notably contributing the empty-element "<empty/>" syntax and the name "XML". Other names that had been put forward for consideration included "MAGMA" (Minimal Architecture for Generalized Markup Applications), "SLIM" (Structured Language for Internet Markup) and "MGML" (Minimal Generalized Markup

Language). The co-editors of the specification were originally Tim Bray and Michael Sperberg-McQueen. Halfway through the project, Bray accepted a consulting engagement with Netscape, provoking vociferous protests from Microsoft. Bray was temporarily asked to resign the editorship. This led to intense dispute in the Working Group, eventually solved by the appointment of Microsoft's Jean Paoli as a third co-editor.

The XML Working Group never met face-to-face; the design was accomplished using a combination of email and weekly teleconferences. The major design decisions were reached in twenty weeks of intense work between July and November of 1996. Further design work continued through 1997, and XML 1.0 became a W3C Recommendation on February 10, 1998.

XML may be viewed as a variant of LISP S-expressions which form tree structures where each node may have its own property list.

Strengths and weaknesses

The features of XML that make it well-suited for data transfer are:

- its simultaneously human- and machine-readable format;
- its support for Unicode, allowing almost any information in any human language to be communicated;
- its ability to represent the most general computer science data structures: records, lists and trees;
- its self-documenting format that describes structure and field names as well as specific values;
- its strict syntax and parsing requirements that allow the necessary parsing algorithms to remain simple, efficient, and consistent.

XML is also heavily used as a format for document storage and processing, both online and offline, and offers several benefits:

- its robust, logically-verifiable format is based on international standards;
- its hierarchical structure is suitable for most (but not all) types of documents;
- it manifests as plain text files, unencumbered by licenses or restrictions;
- it is platform-independent, thus relatively immune to changes in technology;
- it and its predecessor, SGML, have been in use since 1986, so there is extensive experience and software available.

For certain applications, XML also has the following weaknesses:

- Its syntax is fairly verbose and partially redundant. This can hurt human readability and application efficiency, and yields higher storage costs. It can also make XML difficult to apply in cases where bandwidth is limited, though compression can reduce the problem in some cases. This is particularly true for multimedia applications running on cellphones and PDAs which want to use XML to describe images and video.
- Parsers should be designed to recursively handle arbitrarily nested data structures and must perform additional checks to detect improperly formatted or differently ordered syntax or data (this is because the markup is descriptive and partially redundant, as noted above). This causes a significant overhead for most basic uses of XML, particularly where resources may be scarce - for example in embedded systems. Furthermore, additional security considerations arise when XML input is fed from untrusted sources, and resource exhaustion or stack overflows are possible.
- Some consider the syntax to contain a number of obscure, unnecessary features born of its legacy of SGML compatibility. However, an effort to settle on a subset called "Minimal XML" led to the discovery that there was no consensus on *which* features were in fact obscure or unnecessary.
- The basic parsing requirements do not support a very wide array of data types, so interpretation sometimes involves additional work in order to process the desired data from a document. For example, there is no

provision in XML for mandating that "3.14159" is a floating-point number rather than a seven-character string. XML schema languages add this functionality.
- Modelling overlapping (non-hierarchical) data structures requires extra effort.
- Mapping XML to the relational or object oriented paradigms is often cumbersome.
- Some have argued that XML can be used as a data storage only if the file is of low volume, but this is only true given particular assumptions about architecture, data, implementation, and other issues.

Quick syntax tour

Here is an example of a simple recipe expressed using XML:

```
<?xml version="1.0" encoding="UTF-8"?>

<Recipe name="bread" prep_time="5 mins" cook_time="3 hours">
  <title>Basic bread</title>
  <ingredient amount="3" unit="cups">Flour</ingredient>
  <ingredient amount="0.25" unit="ounce">Yeast</ingredient>
  <ingredient amount="1.5" unit="cups">Warm Water</ingredient>
  <ingredient amount="1" unit="teaspoon">Salt</ingredient>
  <Instructions>
   <step>Mix all ingredients together, and knead thoroughly.</step>
   <step>Cover with a cloth, and leave for one hour in warm room.</step>
   <step>Knead again, place in a tin, and then bake in the oven.</step>
  </Instructions>
</Recipe>
```

The first line is the **XML declaration**: it is an optional line stating what version of XML is in use (normally version 1.0), and may also contain information about character encoding and external dependencies.

The remainder of this document consists of nested *elements*, some of which have *attributes* and *content*. An **element** typically consists of two tags, a *start tag* and an *end tag*, possibly surrounding text and other elements. The **start tag** consists of a name surrounded by angle brackets, like "`<step>`"; the **end tag** consists of the same name surrounded by angle brackets, but with a forward slash preceding the name, like "`</step>`". The element's **content** is everything that appears between the start tag and the end tag, including text and other (child) elements. The following is a complete XML element, with start tag, text content, and end tag:

```
<step>Knead again, place in a tin, and then bake in the oven.</step>
```

In addition to content, an element can contain **attributes** — name-value pairs included in the start tag after the element name. Attribute values must always be quoted, using single or double quotes, and each attribute name should appear only once in any element.

```
<ingredient amount="3" unit="cups">Flour</ingredient>
```

In this example, the *ingredient* element has two attributes: *amount*, having value "3", and *units*, having value "cups". In both cases, at the markup level, the names and values of the attributes, just like the names and content of the elements, are just textual data — the "3" and "cups" are not a quantity and unit of measure, respectively, but rather are just character sequences that the document author may be using to *represent* those things.

In addition to text, elements may contain other elements:

```
<Instructions>
 <step>Mix all ingredients together, and knead thoroughly.</step>
 <step>Cover with a cloth, and leave for one hour in warm room.</step>
 <step>Knead again, place in a tin, and then bake in the oven.</step>
</Instructions>
```

In this case, the *Instructions* element contains three *step* elements. XML requires that elements be properly nested — elements may never overlap. For example, this is not well-formed XML, because the *em* and *strong* elements overlap:

```
<!-- WRONG! NOT WELL-FORMED XML! -->
<p>Normal <em>emphasized <strong>strong emphasized</em> strong</strong></p>
```

Every XML document must have exactly one top-level **root element** (alternatively called a *document element*), so the following would also be a malformed XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- WRONG! NOT WELL-FORMED XML! -->
<thing>Thing one</thing>
<thing>Thing two</thing>
```

XML provides special syntax for representing an element with empty content. Instead of writing a start tag followed immediately by an end tag, a document may contain the **empty element tag** where a slash *follows* the element name. The following two examples are exactly equivalent:

```
<foo></foo>
```

```
<foo/>
```

XML provides two methods for escaping (or simply representing) special characters: *entity references* and *numeric character references*. An **entity** in XML is a named body of data, usually representing text, such as an unusual character. An **entity reference** is a placeholder for that entity, and consists of the entity's name preceded by an ampersand ("&") and followed by a semicolon (";"). XML has several predeclared entities, such as "lt" (referenced as "<") for the left angle bracket (<) and "amp" (referenced as "&") for the ampersand (&) itself, and it is possible to declare additional ones if desired. Aside from representing individual characters, reproducing chunks of boilerplate text is another common use for entities. Here is an example using a predeclared XML entity to escape the ampersand in the name "AT&T":

```
<company-name>AT&amp;T</company-name>
```

The full list of predeclared entities are

- & (&)
- < (<)
- > (>)
- ' (')
- " (")

If more entities need to be declared, this is done in the document's *DTD*, which is not demonstrated in this example, for brevity.

Numeric **character references** look like entities, but instead of a name, they contain the "#" character followed by a number between the ampersand and the semicolon. The number (in decimal or hexadecimal) represents a Unicode code point, and is typically used to represent characters that are not easily encodable, such as an Arabic character in a document produced on a European computer. The ampersand in the "AT&T" example could also be escaped like this (decimal 38 is the Unicode value for "&"):

```
<company-name>AT&#038;T</company-name>
```

There are many more rules necessary to be sure of writing well-formed XML documents, such as the exact characters allowed in an XML name, but this quick tour provides the basics necessary to read and understand many XML documents.

Correctness in an XML document

For an XML document to be correct, it must be:

- **Well-formed.** A well-formed document conforms to all of XML's syntax rules. For example, if a non-empty element has an opening tag with no closing tag, it is not *well-formed*. A document that is not well-formed is not considered to be XML; a parser is required to refuse to process it.
- **Valid.** A valid document has data that conforms to a particular set of user-defined content rules that describe correct data values and locations. For example, if an element in a document is required to contain text that can be interpreted as being an integer numeric value, and it instead has the text "hello", is empty, or has other elements in its content, then the document is not *valid*.

Well-formed documents

An XML document is text, which is a sequence of characters. The specification **requires** support for Unicode encodings UTF-8 and UTF-16 (UTF-32 is not mandatory). The use of other non-Unicode based encodings, such as ISO-8859, is admitted and is indeed widely used and supported.

A well-formed document must conform to the following rules, among others:

- One and only one root element exists for the document. However, the XML declaration, processing instructions, and comments can precede the root element.
- Non-empty elements are delimited by both a start-tag and an end-tag.
- Empty elements may be marked with an empty-element (self-closing) tag, such as `<IAmEmpty/>`. This is equal to `<IAmEmpty></IAmEmpty>`.
- All attribute values are quoted, either single (') or double (") quotes. Single quotes close a single quote and double quotes close a double quote.
- Tags may be nested but may not overlap. Each non-root element must be completely contained in another element.
- The document complies to its character set definition. The charset is usually defined in the xml declaration but it can be provided by the transport protocol, such as HTTP. If no charset is defined, usage of a Unicode encoding is assumed, defined by the Unicode Byte Order Mark. If the mark does not exist, UTF-8 is the default.

Element names are case-sensitive. For example, the following is a well-formed matching pair

```
<Step> ... </Step>
```

whereas this is not

```
<Step> ... </step>
```

The careful choice of names for XML elements will convey the meaning of the data in the markup. This increases human readability while retaining the rigor needed for software parsing.

Choosing meaningful names implies the semantics of elements and attributes to a human reader without reference to external documentation. However, this can lead to verbosity, which complicates authoring and increases file size.

Valid documents

An XML document that complies with a particular schema, in addition to being well-formed, is said to be **valid**.

An XML schema is a description of a type of XML document, typically expressed in terms of constraints on the structure and content of documents of that type, above and beyond the basic constraints imposed by XML itself. A number of standard and proprietary XML schema languages have emerged for the purpose of formally expressing such schemas, and some of these languages are XML-based, themselves.

Before the advent of generalised data description languages such as SGML and XML, software designers had to define special file formats or small languages to share data between programs. This required writing detailed specifications and special-purpose parsers and writers.

XML's regular structure and strict parsing rules allow software designers to leave parsing to standard tools, and since XML provides a general, data model-oriented framework for the development of application-specific languages, software designers need only concentrate on the development of rules for their data, at relatively high levels of abstraction.

Well-tested tools exist to validate an XML document "against" a schema: the tool automatically verifies whether the document conforms to constraints expressed in the schema. Some of these validation tools are included in XML parsers, and some are packaged separately.

Other usages of schemas exist: XML editors, for instance, can use schemas to support the editing process.

DTD

The oldest schema format for XML is the Document Type Definition (DTD), inherited from SGML. While DTD support is ubiquitous due to its inclusion in the XML 1.0 standard, it is seen as limited for the following reasons:

- It has no support for newer features of XML, most importantly namespaces.
- It lacks expressivity. Certain formal aspects of an XML document cannot be captured in a DTD.
- It uses a custom non-XML syntax, inherited from SGML, to describe the schema.

XML Schema

A newer XML schema language, described by the W3C as the successor of DTDs, is XML Schema, or more informally referred to in terms of the initialism for XML Schema instances, XSD (XML Schema Definition). XSDs are far more powerful than DTDs in describing XML languages. They use a rich datatyping system, allow for more detailed constraints on an XML document's logical structure, and are required to be processed in a more robust validation framework. Additionally, XSDs use an XML based format, which makes it possible to use ordinary XML tools to help process them, although WXS (W3C XML Schema) implementations require much more than just the ability to read XML.

Criticisms of WXS include the following:

- The specification is very large, which makes it difficult to understand and implement.
- The XML-based syntax leads to verbosity in schema description, which makes XSDs harder to read and write.

RELAX NG

Another popular schema language for XML is RELAX NG. Initially specified by OASIS, RELAX NG is now also an ISO international standard (as part of DSDL). It has two formats: an XML based syntax and a non-XML compact syntax. The compact syntax aims to increase readability and writability, but since there is a well-defined way to translate compact syntax to the XML syntax and back again by means of James Clark's Trang conversion tool, the advantage of using standard XML tools is not lost. Compared to XML Schema, RELAX NG has a simpler definition and validation framework, making it easier to use and implement. It also has the ability to use any datatype framework on a plug-in basis; for example, a RELAX NG schema author can require values in an XML document to conform to definitions in XML Schema Datatypes.

Other schema languages

Some schema languages not only describe the structure of a particular XML format but also offer limited facilities to influence processing of individual XML files that conform to this format. DTDs and XSDs both have this ability; they can for instance provide attribute defaults. RELAX NG intentionally does not provide these facilities.

Displaying XML on the web

Extensible Stylesheet Language (XSL) is a supporting technology that describes how to format or transform the data in an XML document. The document is changed to a format suitable for browser display. The process is similar to applying a CSS to an HTML document for rendering.

Without using CSS or XSL, a generic XML document is rendered differently in different web browsers. Browsers like Internet Explorer and Mozilla allow viewing of generic XML document similar to directory structure, e.g. expanding or collapsing a subtree. In order to allow CSS styling, the XML document must include a special reference to a style sheet:

```
<?xml-stylesheet type="text/css" href="myStyleSheet.css"?>
```

This is different from the HTML way to apply a stylesheet, which uses the `<link>` element.

For XSL Transformations (XSLT), this is also very similar:

```
<?xml-stylesheet type="text/xsl" href="transform.xsl"?>
```

Client-side XSLT is not supported in Opera.

While browser-based XML rendering develops, the alternative is conversion into HTML or PDF or other formats *on the server*. Programs like Cocoon process an XML file against a stylesheet (and can perform other processing as well) and send the output back to the user's browser without the user needing to be aware of what has been going on in the background.

XML extensions

- **XPath** It is possible to refer to individual components of an XML document using XPath. This allows stylesheets in (for example) XSL and XSLT to dynamically "cherry-pick" pieces of a document in any sequence needed in order to compose the required output.
- **XQuery** is to XML what SQL is to relational databases.
- **XML namespaces** enable the same document to contain XML elements and attributes taken from different vocabularies, without any naming collisions occurring.
- **XML Signature** defines the syntax and processing rules for creating digital signatures on XML content.
- **XML Encryption** defines the syntax and processing rules for encrypting XML content.
- **XPointer** is a system for addressing components of XML based internet media.

Processing XML files

SAX and DOM are APIs widely used to process XML data. SAX is used for serial processing whereas DOM is used for random-access processing. Another form of XML Processing API is data binding, where XML data is made available as a strongly typed programming language data structure, in contrast to the DOM. Example data binding systems are the Java Architecture for XML Binding (JAXB) [1] (http://java.sun.com/xml/jaxb/) and the Strathclyde Novel Architecture for Querying XML (SNAQue) [2] (http://www.cis.strath.ac.uk/research/snaque/).

A filter in the Extensible Stylesheet Language (XSL) family can transform an XML file for displaying or printing.

- **XSL-FO** is a declarative, XML-based page layout language. An XSL-FO processor can be used to convert an XSL-FO document into another non-XML format, such as PDF.
- **XSLT** is a declarative, XML-based document transformation language. An XSLT processor can use an XSLT *stylesheet* as a guide for the conversion of the data tree represented by one XML document into another tree that can then be serialized as XML, HTML, plain text, or any other format supported by the processor.
- **XQuery** is a W3C language for querying, constructing and transforming XML data.
- **XPath** is a path expression language for selecting data within an XML file. XPath is a sublanguage of both XQuery and XSLT.

The native file format of OpenOffice.org and AbiWord is XML. Some parts of Microsoft Office 11 will also be able to edit XML files with a user-supplied schema (but not a DTD), and on June 2, 2005 Microsoft announced that, by late 2006 all the files created by users of its Office suite of software will be formatted with web-centered XML specifications. There are dozens of other XML editors available.

Versions of XML

There are two current versions of XML. The first, *XML 1.0*, was initially defined in 1998. It has undergone minor revisions since then, without being given a new version number, and is currently in its third edition, as published on February 4, 2004. It is widely implemented and still recommended for general use. The second, *XML 1.1*, was initially published on the same day as XML 1.0 Third Edition. It contains features — some contentious — that are

intended to make XML easier to use for certain classes of users (mainframe programmers, mainly). XML 1.1 is not very widely implemented and is recommended for use only by those who need its unique features.

XML 1.0 and XML 1.1 differ in the requirements of characters used for element names, attribute names etc.: XML 1.0 only allows characters which are defined in Unicode 2.0, which includes most world scripts, but excludes scripts which only entered in a later Unicode version, such as Mongolian, Cambodian, Amharic, Burmese, etc.. XML 1.1 only *disallows* certain control characters, which means that any other character can be used, even if it is not defined in the current version of Unicode.

It should be noted here that the restriction present in XML 1.0 only applies to element/attribute names: both XML 1.0 and XML 1.1 allow for the use of full Unicode in the content itself. Thus XML 1.1 is only needed if in addition to using a script added after Unicode 2.0 you also wish to write element and attribute names in that script.

Other minor changes between XML 1.0 and XML 1.1 are that control characters are now allowed to be included but only when escaped, and two special Unicode line break characters are included, which must be treated as whitespace.

XML 1.0 documents are well-formed XML 1.1 documents with one exception: XML documents that contain unescaped C1 control characters are now malformed: this is because XML 1.1 requires the C1 control characters to be escaped with numeric character references.

There are also discussions on an XML 2.0, although it remains to be seen if such will ever come about. XML-SW (SW for skunk works), written by one of the original developers of XML, contains some proposals for what an XML 2.0 might look like: elimination of DTDs from syntax, integration of namespaces, XML Base and XML Information Set (*Infoset*) into the base standard.

The World Wide Web Consortium also has a XML Binary Characterization Working Group doing preliminary research into use cases and properties for a binary encoding of the XML infoset. The working group is not chartered to produce any official standards. Since XML is by definition text-based, ITU-T and ISO are using the name *Fast Infoset (http://asn1.elibel.tm.fr/xml/finf.htm)* for their own binary infoset to avoid confusion (see ITU-T Rec. X.891 | ISO/IEC 24824-1).

See also

- XML schema languages: DTD, XML Schema, RELAX NG, DSDL
- XML processing APIs: DOM, SAX, E4X
- CSS, XSL
- S-expression, XML query language
- Abstract syntax tree (AST)
- XRI, XDI
- Extensible Metadata Platform (XMP), used in graphics applications
- SVG
- ASN.1
- WBXML
- Serialization
- List of general purpose markup languages
- Comparison of general purpose markup languages
- Comparison of layout engines (XML)
- Single source publishing
- Extensible Binary Meta Language

References

- Bray, Tim (February 2005). "A conversation with Tim Bray: Searching for ways to tame the world's vast stores of information." *Association for Computing Machinery's Queue*. 3 (1). link (http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=282)

External links

Specifications

- World Wide Web Consortium XML homepage (http://www.w3.org/XML/)
- The XML 1.0 specification (http://www.w3.org/TR/REC-xml)
- The XML 1.1 specification (http://www.w3.org/TR/xml11)

Basic Readings

- The XML FAQ (http://xml.silmaril.ie/)
- Annotated XML Specification (http://www.xml.com/axml/testaxml.htm)

Web-Zines

- XML.com (http://www.xml.com/)

XML Editors

- AutoXML (http://autoxml.atspace.com/) Text-based editor with limited XML-awareness. Freeware.
- EditiX (http://www.editix.com/) Commercial.
- OpenXml Editor (http://www.philo.de/xmledit) Text-based editor. Open source (Delphi).
- XmlSpy. (http://www.xmlspy.com/) Commercial.

Retrieved from "http://en.wikipedia.org/wiki/XML"

Categories: XML | Markup languages | W3C standards | Technical communication | Computer file formats | Abbreviations